CS 371D
Distributed Computing

Lorenzo Alvisi
Fangkai Yang

What is a distributed system?

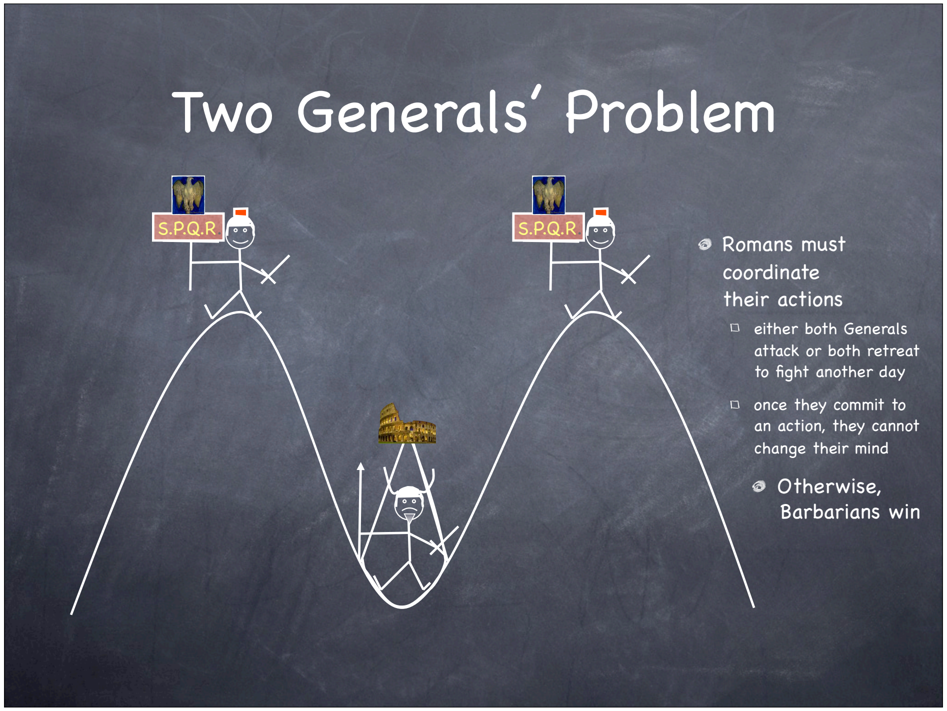
"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

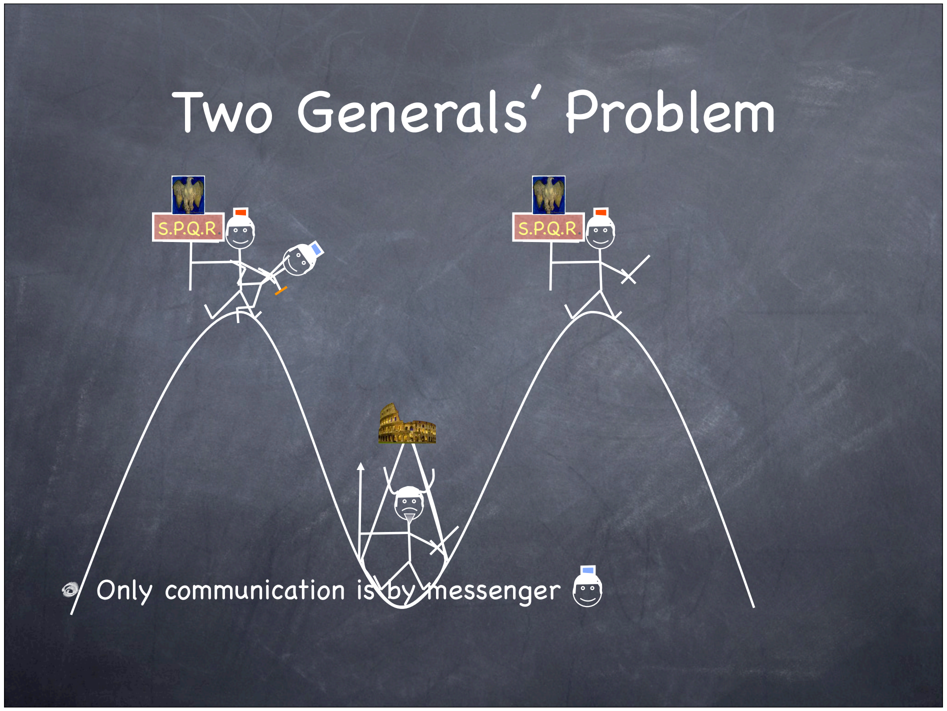
Leslie Lamport

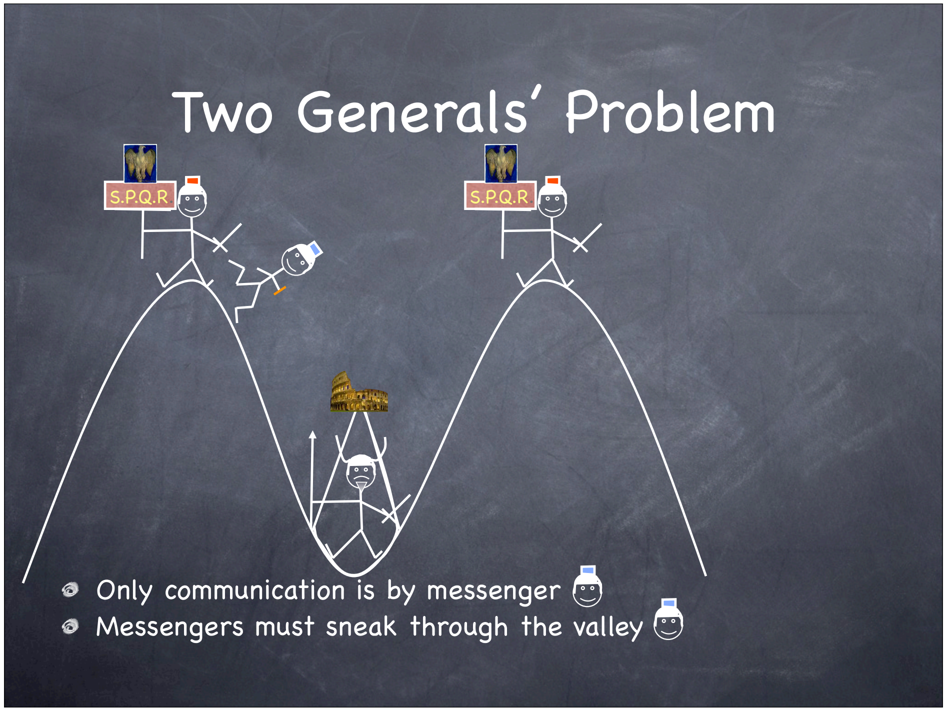
A few intriguing questions

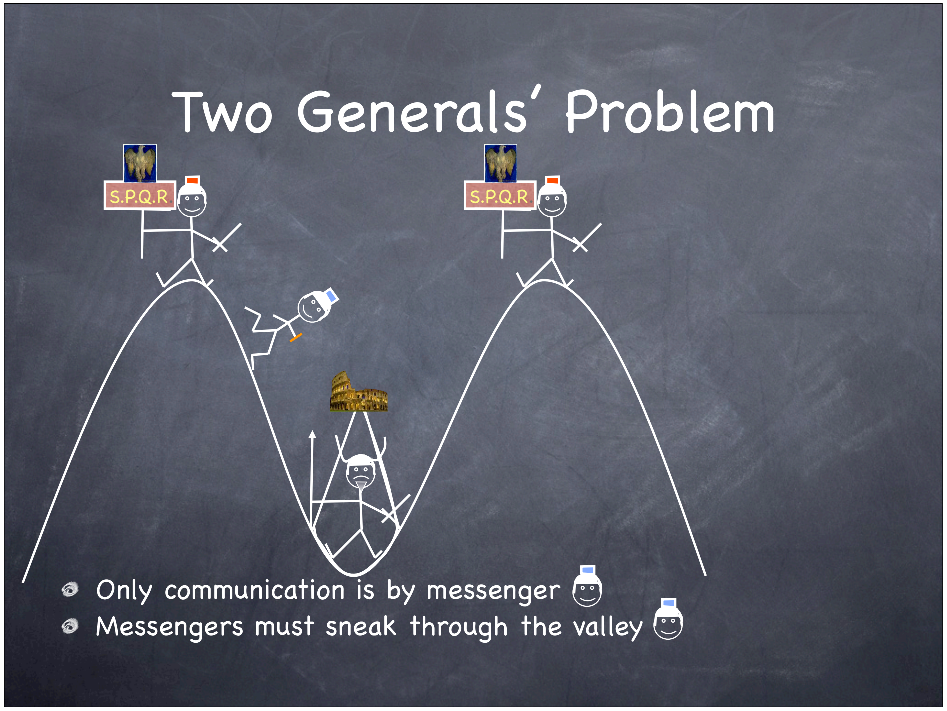
- How do we talk about a distributed execution?
- Can we draw global conclusions from local information?
- Can we coordinate operations without relying on synchrony?
- For the problems we know how to solve, how do we characterize the "goodness" of our solution?
- Are there problems that simply cannot be solved?
- What are useful notions of consistency, and how do we maintain them?
- What if part of the system is down? Can we still do useful work? What if instead part of the system becomes "possessed" and starts behaving arbitrarily–all bets are off?

Saving the world before bedtime

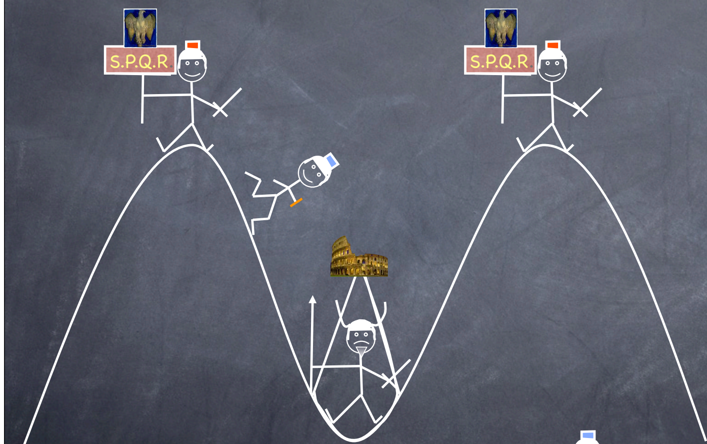

Two Generals' Problem
S.P.Q.R
S.P.Q.R
Romans must coordinate their actions
either both Generals attack or both retreat to fight another day
once they commit to an action, they cannot change their mind
Otherwise, Barbarians win


Two Generals' Problem
S.P.Q.R
S.P.Q.R
Only communication is by messenger


Two Generals' Problem
S.P.Q.R
S.P.Q.R
Only communication is by messenger
Messengers must sneak through the valley

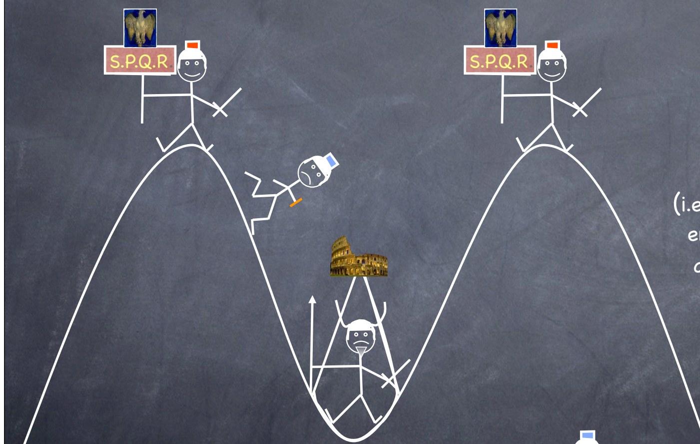

Two Generals' Problem
S.P.Q.R
S.P.Q.R
Only communication is by messenger
Messengers must sneak through the valley

Two Generals' Problem



- Only communication is by messenger
- Messengers must sneak through the valley
- They don't always make it

Two Generals' Problem



Problem:
Save Western Civilization
(i.e. design a protocol that ensures Romans always attack simultaneously)

- Only communication is by messenger
- Messengers must sneak through the valley
- They don't always make it

Two General's Problem

Claim: There is no non-trivial protocol that guarantees that the Romans will always attack simultaneously

Two General's Problem

Claim: There is no non-trivial protocol that guarantees that the Romans will always attack simultaneously

Proof: By contradiction

- Let n be the smallest number of messages needed by a solution
- Consider the n-th message m_{last}
  - The state of the sender of m_{last} cannot depend on the receipt of m_{last}
  - The state of the receiver of m_{last} cannot depend on the receipt of m_{last} because in some executions m_{last} could be lost
  - So both sender and receiver would come to the same conclusion even without sending m_{last}
  - We now have a solution requiring only $n-1$ messages - but n was supposed to be the smallest number of messages!

Two General's Problem

Claim: There is no non-trivial protocol that guarantees that the Romans will always attack simultaneously

Proof: By contradiction

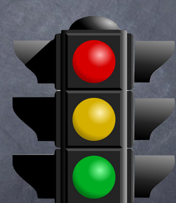
- Let n be the smallest number of messages needed by a solution
- Consider the n-th message m_{last}
  - The state of the sender of m_{last} cannot depend on the receipt of m_{last}
  - The state of the receiver of m_{last} cannot depend on the receipt of m_{last} because in some executions m_{last} could be lost
  - So both sender and receiver would come to the same conclusion even without sending m_{last}
  - We now have a solution requiring only $n-1$ messages – but n was supposed to be the smallest number of messages! Contradiction

If only I had known...

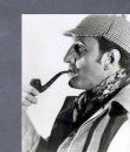
- Solving the Two Generals Problem requires common knowledge
  - "everyone knows that everyone knows that everyone knows..." – you get the picture
- Alas...
  - Common knowledge cannot be achieved by communicating through unreliable channels

Do you trust traffic lights?

- Suppose each driver is told:
  - RED means "Stop"
  - GREEN means "Go"
  - Follow the rules!
- Do you feel safe driving?



The Case of the Muddy Children



The Case of the Muddy Children

- n children go playing
- Children are truthful, perceptive, intelligent
- Mom says: "Don't get muddy!"
- A bunch (say, k) get mud on their forehead
- Daddy comes, looks around, and says:
- "Some of you got a muddy forehead!"

The Case of the Muddy Children

- n children go playing
- Children are truthful, perceptive, intelligent
- Mom says: "Don't get muddy!"
- A bunch (say, k) get mud on their forehead
- Daddy comes, looks around, and says:
  - "Some of you got a muddy forehead!"
- Dad then asks repeatedly:
  - "Do you know whether you have mud on your own forehead?"
- What happens?

Elementary...

- Claim: The first $k-1$ times the father asks, all children will reply "No", but the k-th time all dirty children with reply yes
- Proof: By induction on k
  - $k=1$
    The child with the muddy forehead sees no one else dirty. Dad says someone is, so he must be the one
  - $k=2$ - Two muddy children, a and b .
    - Each answers "No" the first time because it sees the other
    - When a sees b say No, she realizes she must be dirty, because b must have seen a dirty child, and a sees no one dirty but b. So a must be dirty!
  - $k=3$ - Three muddy children, a, b, and c...

Elementary?

- Suppose $k > 1$
- Every one knows that someone has a dirty forehead before Dad announces it...
- Does Daddy still need to speak up?

Elementary?

- Suppose $k > 1$
- Every one knows that someone has a dirty forehead before Dad announces it...
- Does Daddy still need to speak up?
- Claim: Unless he does, the muddy children will never be able to determine that their forehead are muddy!

Common Knowledge: The Revenge

- Let p = "Someone's forehead is dirty"
- Every one knows p
- But, unless the father speak, if $k = 2$ not every one knows that everyone knows p!
    - Suppose a and b are dirty. Before the father speaks a does not know whether b knows p
- If $k = 3$, not every one knows that every one knows that every one knows p ...

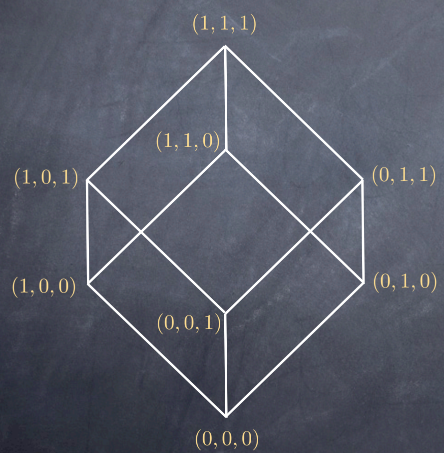
Would it work if...

- ... the father took every child aside and told them individually (without others noticing) that someone's forehead is muddy?

Would it work if...

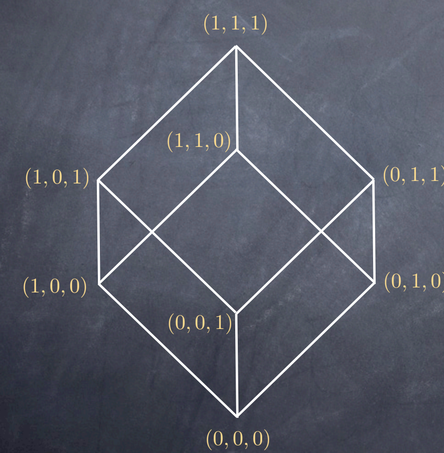
- ... the father took every child aside and told them individually (without others noticing) that someone's forehead is muddy?
- ... every child had (unknown to the other children) put a miniature microphone on every other child so they can hear what the father says in private to them?

Parallel Worlds!



- k = 3
- Each node labeled with a tuple that represents a possible world: (1, 0, 1) is a world where only child 2 does not have a muddy forehead

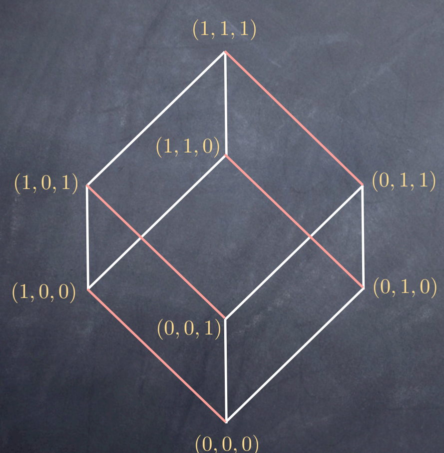
Parallel Worlds!



- k = 3
- Each node labeled with a tuple that represents a possible world: (1, 0, 1) is a world where only child 2 does not have a muddy forehead
- Each edge is labeled by the color of the child for which the two endpoints are both possible worlds

Child 1
Child 2
Child 3

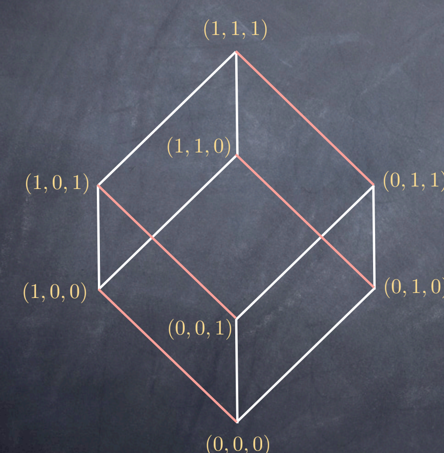
Parallel Worlds!



- k = 3
- Each node labeled with a tuple that represents a possible world: (1, 0, 1) is a world where only child 2 does not have a muddy forehead
- Each edge is labeled by the color of the child for which the two endpoints are both possible worlds

Child 1
Child 2
Child 3

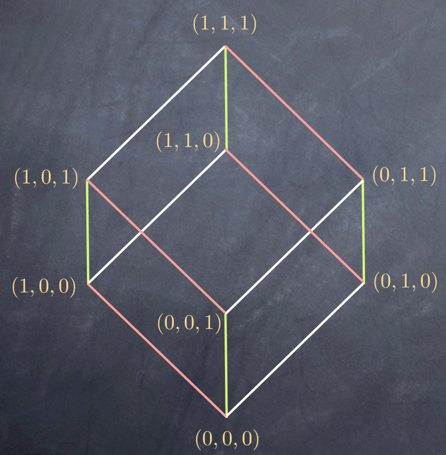
Parallel Worlds!



- k = 3
- Each node labeled with a tuple that represents a possible world: (1, 0, 1) is a world where only child 2 does not have a muddy forehead
- Each edge is labeled by the color of the child for which the two endpoints are both possible worlds

Child 1
Child 2
Child 3

Parallel Worlds!



$k = 3$

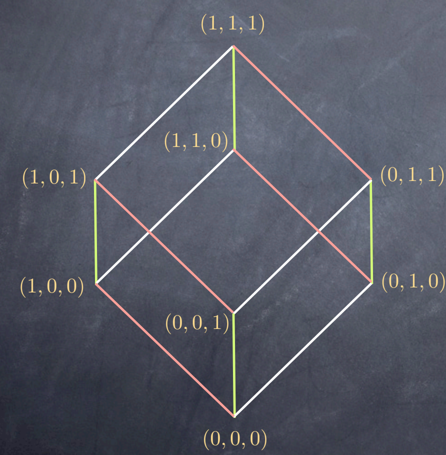
- Each node labeled with a tuple that represents a possible world: (1, 0, 1) is a world where only child 2 does not have a muddy forehead
- Each edge is labeled by the color of the child for which the two endpoints are both possible worlds

Child 1

Child 2

Child 3

After the father speaks



$k = 3$

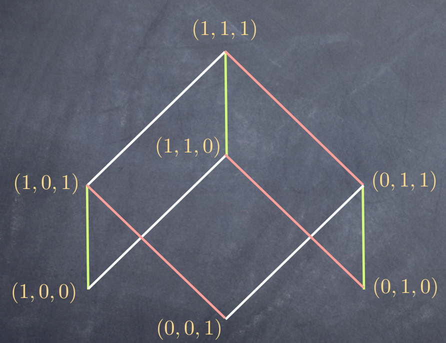
- The state $(0, 0, 0)$ becomes impossible
- All the edges that depart from it are eliminated

Child 1

Child 2

Child 3

If everyone answers "No" to the 1st question..



- All states with a single 1 become impossible!
- All the edges that depart from them are eliminated

Child 1

Child 2

Child 3

Much more...

- There is an entire logic that formalizes what knowledge participants acquire while running a protocol
  - J. Halpern and Y. Moses
    Knowledge and Common Knowledge in a Distributed Environment
    E.W. Dijkstra Prize 2009.

Global Predicate Detection and Event Ordering

Our Problem

To compute predicates
over the state of
a distributed application

Model

- Message passing
- No failures
- Two possible timing assumptions:
  1. Synchronous System
  2. Asynchronous System
     - No upper bound on message delivery time
     - No bound on relative process speeds

Asynchronous systems

- Weakest possible assumptions
  - cfr. "finite progress axiom"
- Weak assumptions $\equiv$ less vulnerabilities
- Asynchronous $\neq$ slow
- "Interesting" model w.r.t. failures (ah ah ah!)

Client-Server

Processes exchange messages using
Remote Procedure Call (RPC)

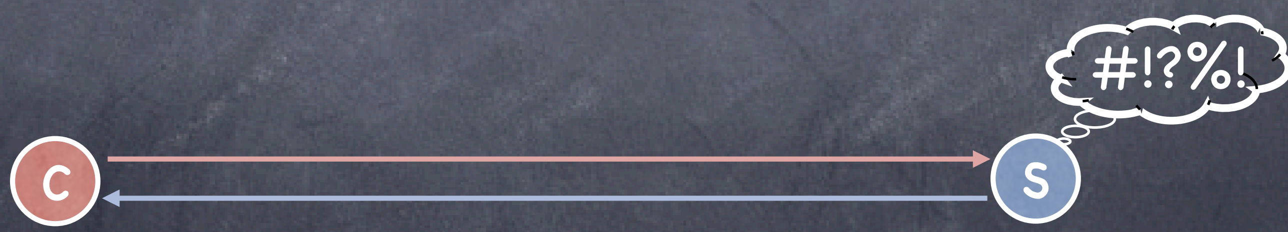
A client requests a service by sending the server a message. The client blocks while waiting for a response

Client-Server

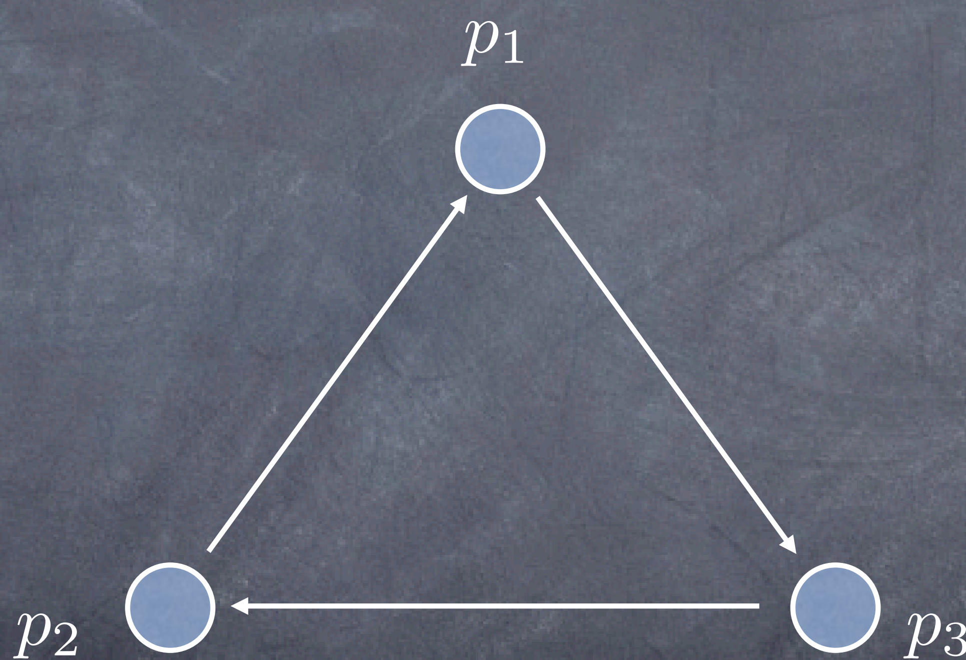
Processes exchange messages using Remote Procedure Call (RPC)

A client requests a service by sending the server a message. The client blocks while waiting for a response

The server computes the response (possibly asking other servers) and returns it to the client



Deadlock!



Goal

Design a protocol by which a processor can determine whether a global predicate (say, deadlock) holds

Wait-For Graphs

- Draw arrow from p_i to p_j if p_j has received a request but has not responded yet

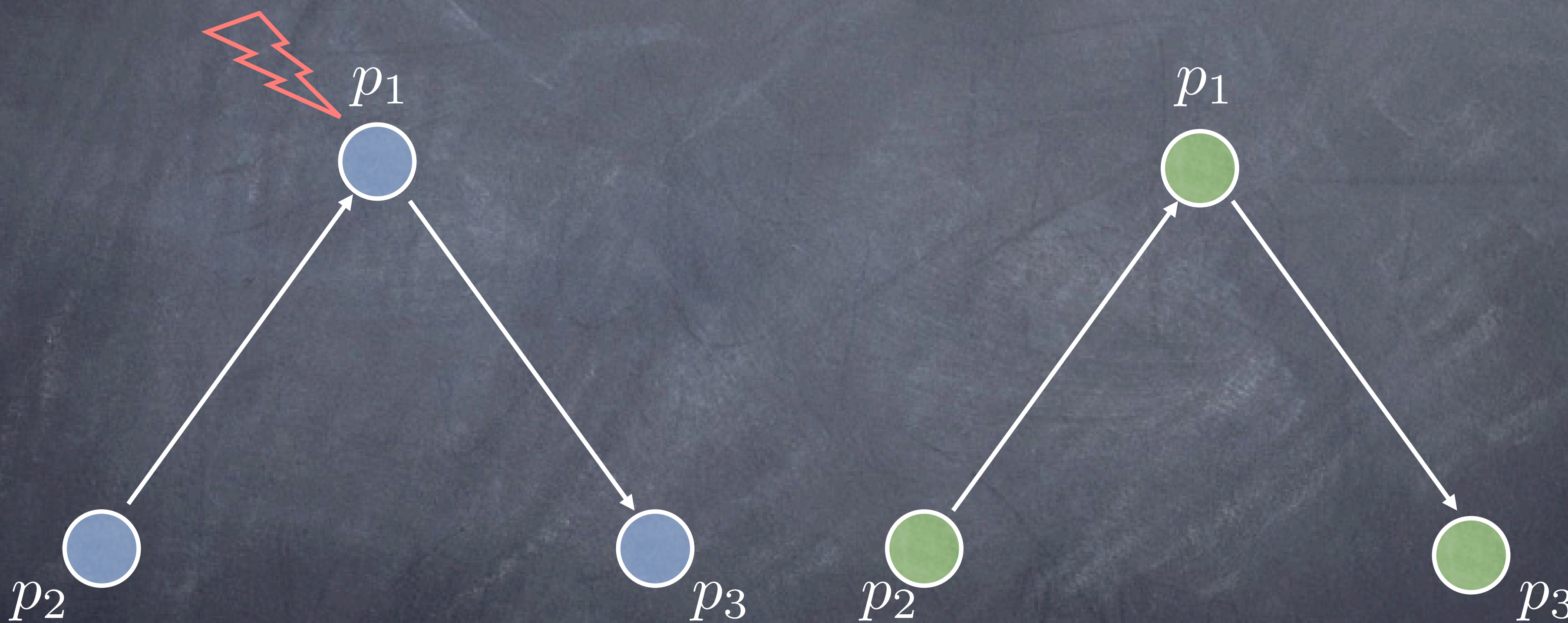
Wait-For Graphs

- Draw arrow from p_i to p_j if p_j has received a request but has not responded yet
- Cycle in WFG $\Rightarrow$ deadlock
- Deadlock $\Rightarrow \Diamond$ cycle in WFG

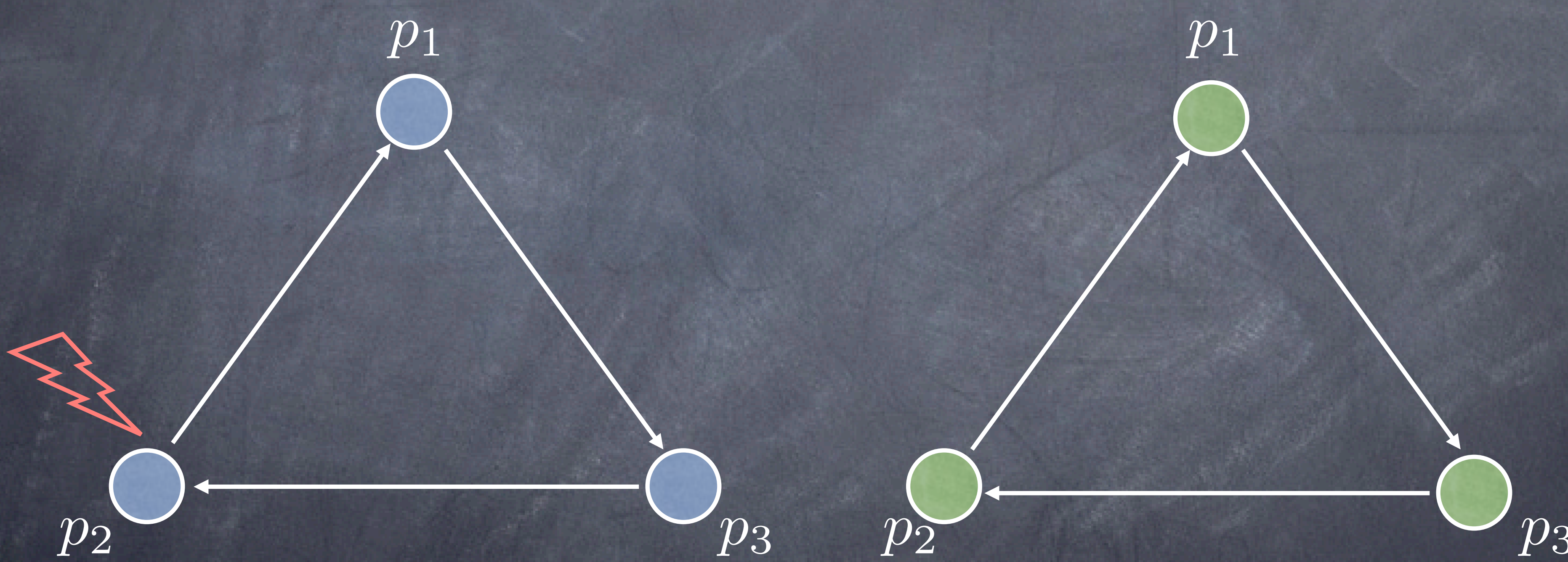
The protocol

- p_0 sends a message to $p_1 \ldots p_3$
- On receipt of p_0's message, p_i replies with its state and wait-for info

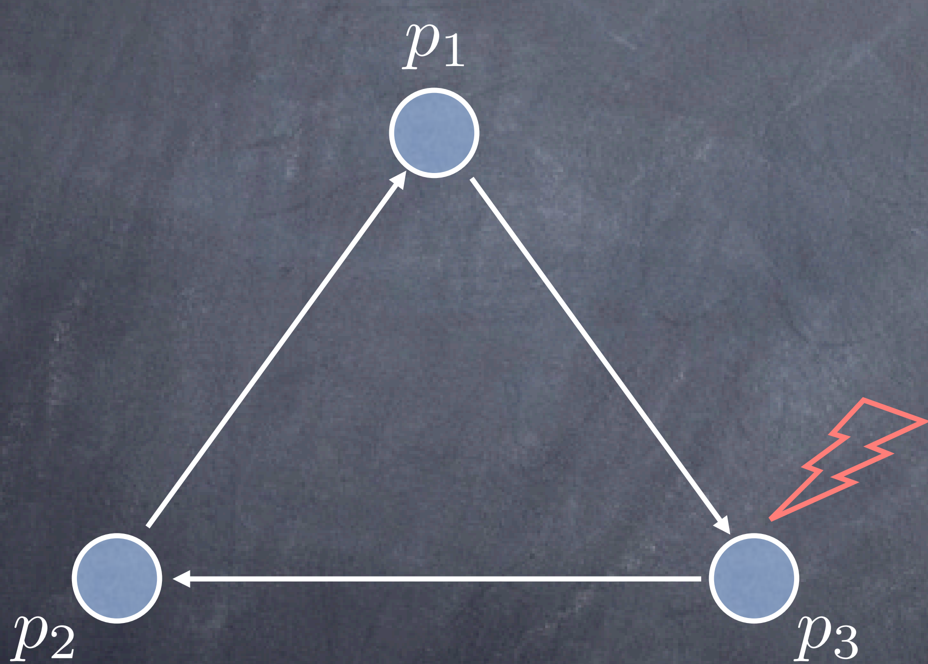
An execution

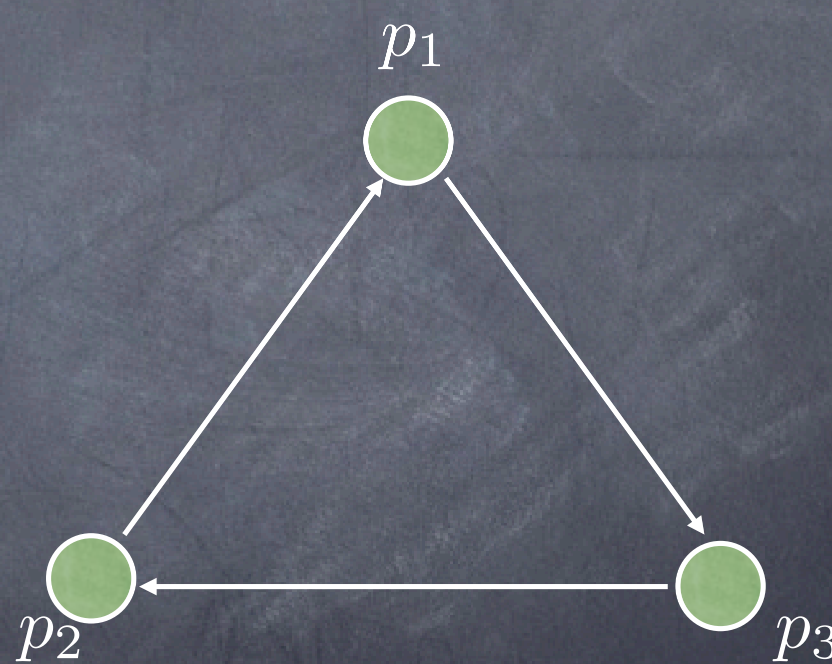


An execution



An execution





Ghost Deadlock!

Houston,
we have a problem...

- Asynchronous system
  - no centralized clock, etc. etc.
- Synchrony useful to
  - coordinate actions
  - order events
- Mmmmhhh...

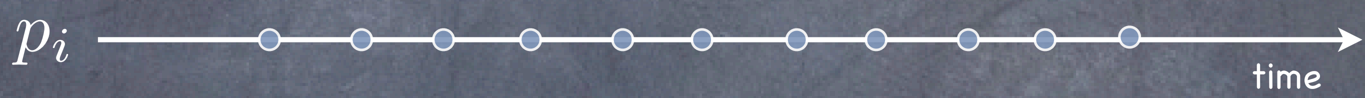
Events and Histories

- Processes execute sequences of events
- Events can be of 3 types: local, send, and receive
- e_p^i is the i-th event of process p
- The local history h_p of process p is the sequence of events executed by process p
  - h_p^k : prefix that contains first k events
  - h_p^0 : initial, empty sequence
- The history H is the set $h_{p_0} \cup h_{p_1} \cup \ldots h_{p_{n-1}}$

NOTE: In H, local histories are interpreted as sets, rather than sequences, of events

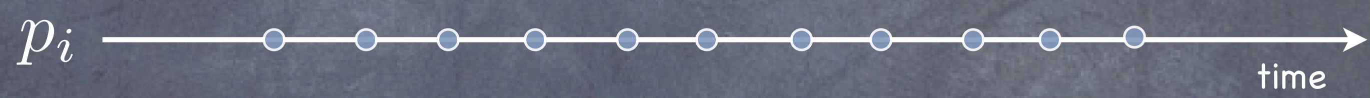
Ordering events

- Observation 1:
  - Events in a local history are totally ordered

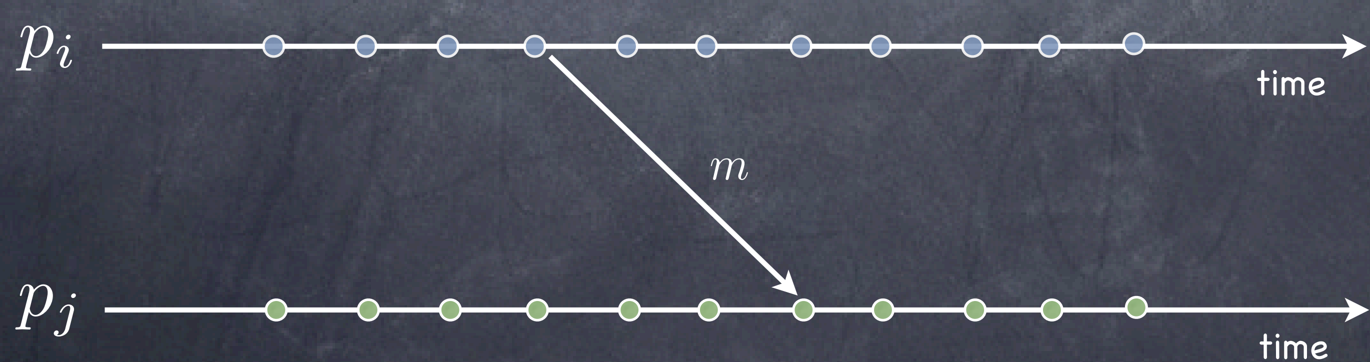


Ordering events

- Observation 1:
  - Events in a local history are totally ordered



- Observation 2:
  - For every message m, $send(m)$ precedes $receive(m)$



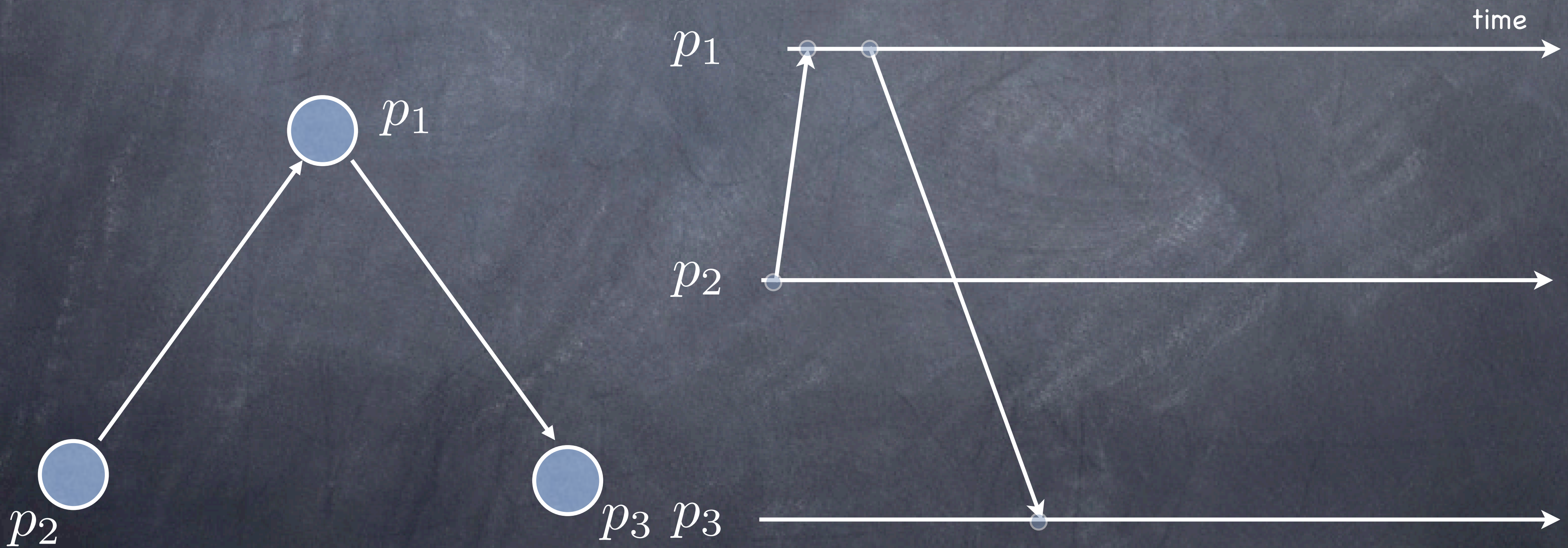
Happened-before (Lamport[1978])

A binary relation $\rightarrow$ defined over events

1. if $e_i^k, e_i^l \in h_i$ and $k < l$, then $e_i^k \rightarrow e_i^l$
2. if $e_i = send(m)$ and $e_j = receive(m)$, then $e_i \rightarrow e_j$
3. if $e \rightarrow e'$ and $e' \rightarrow e''$ then $e \rightarrow e''$

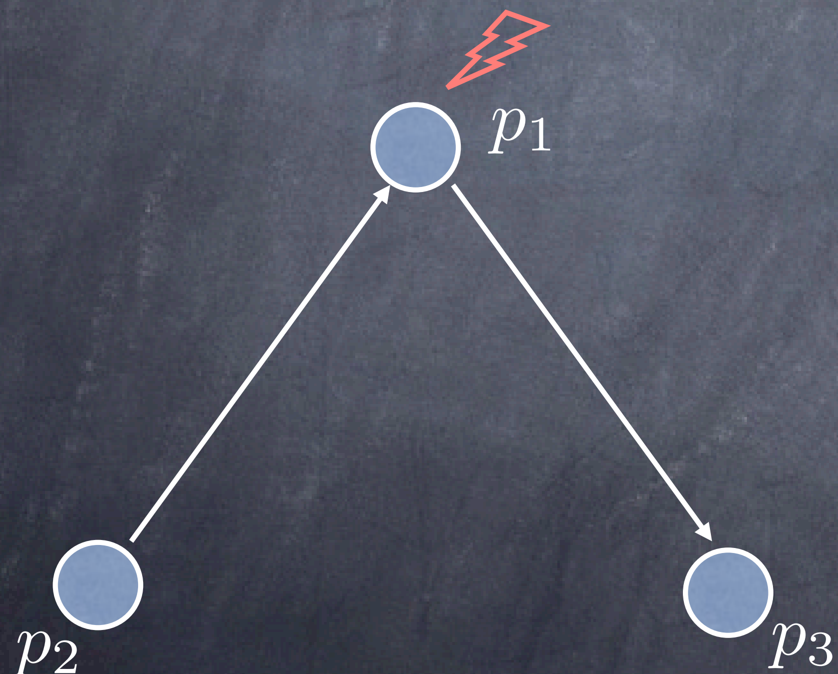
Space-Time diagrams

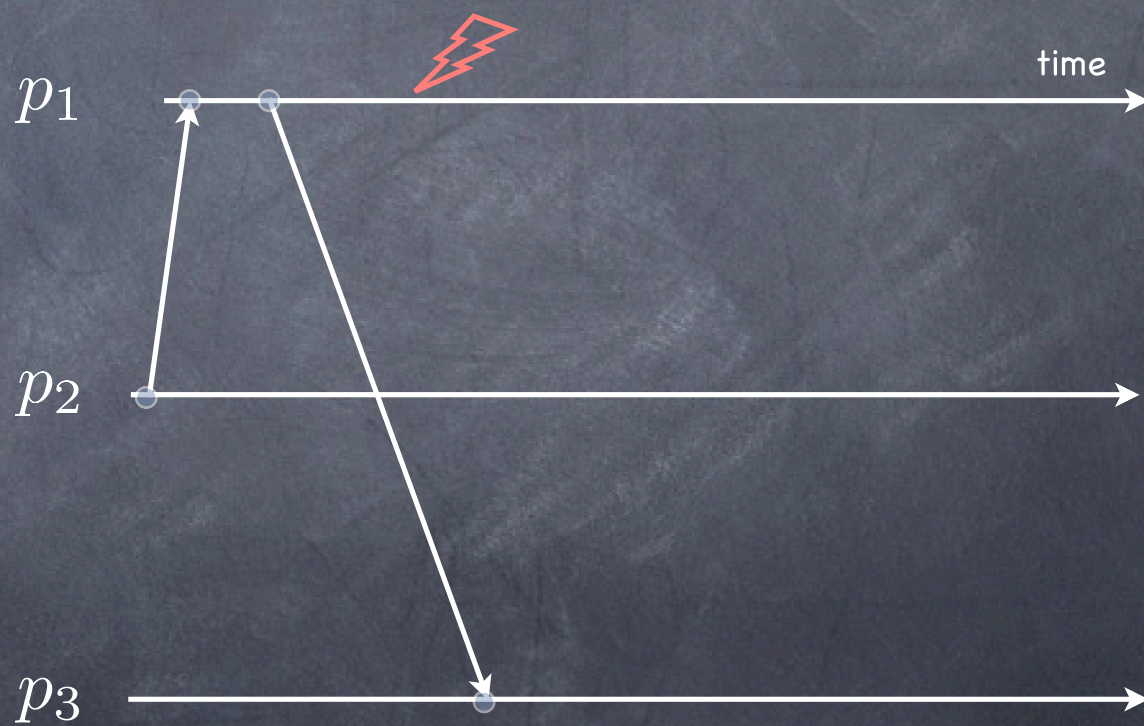
A graphic representation of a distributed execution



Space-Time diagrams

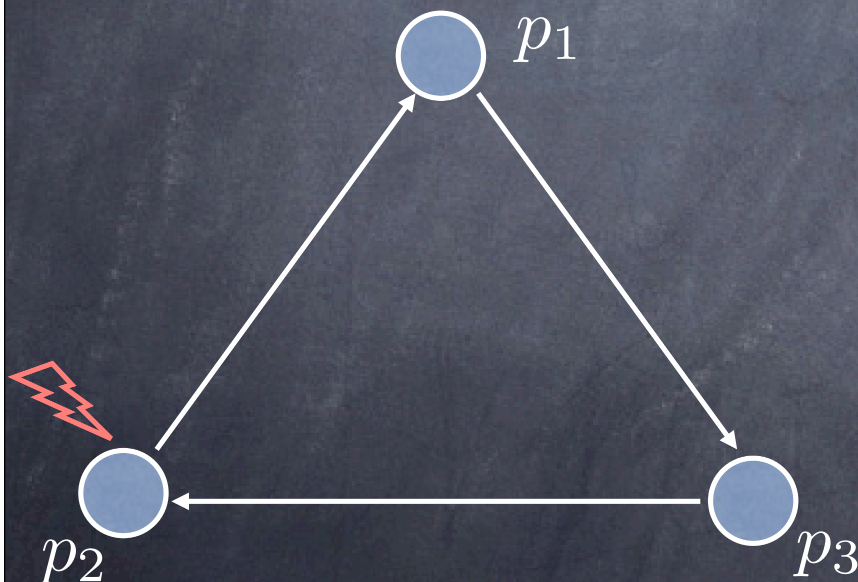
A graphic representation of a distributed execution

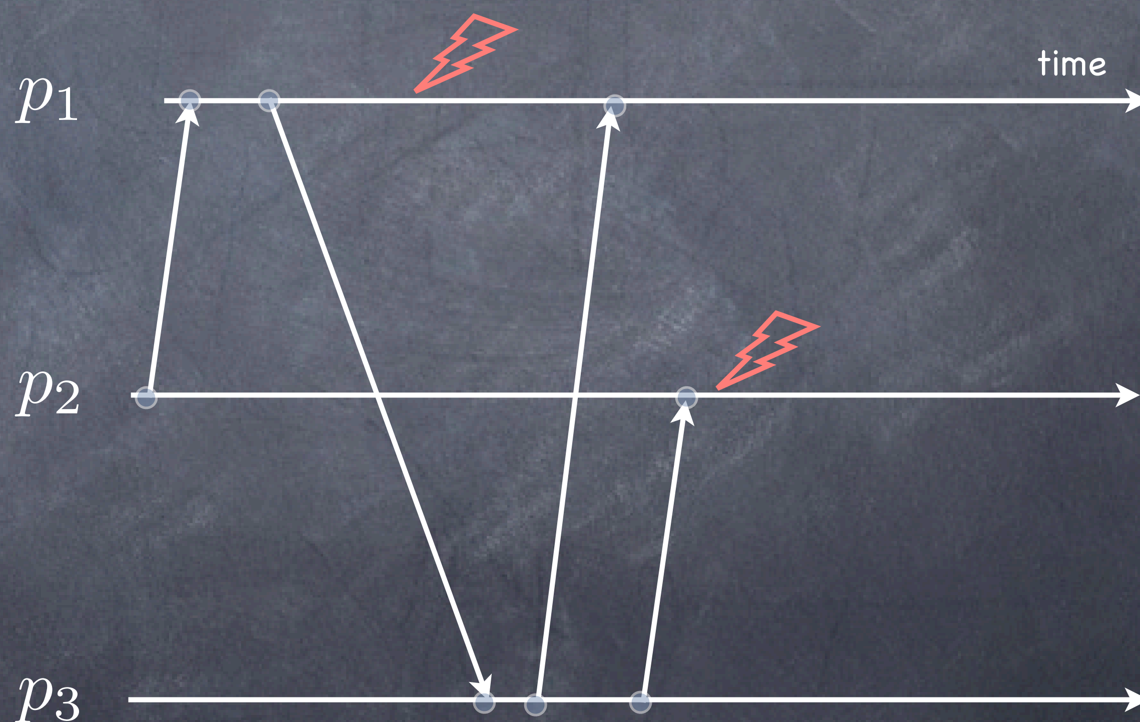




Space-Time diagrams

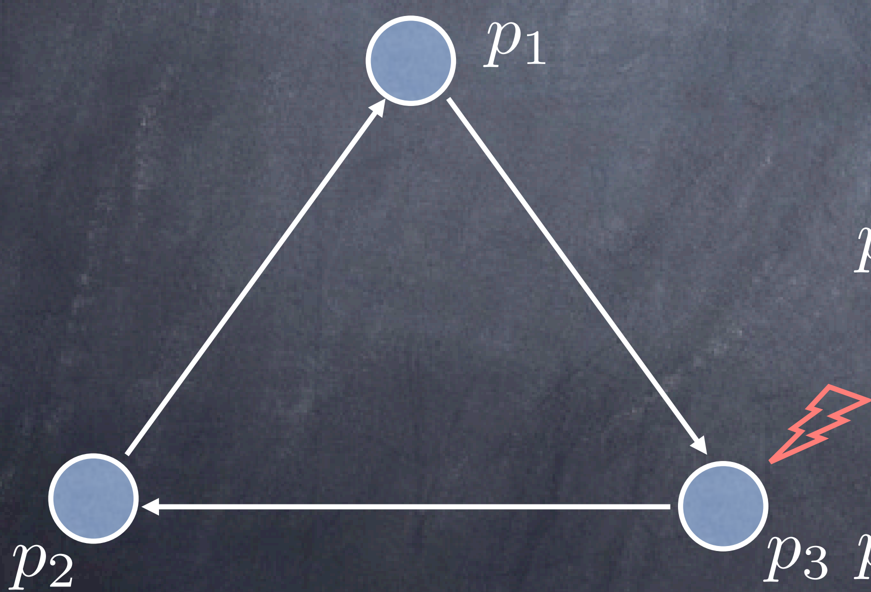
A graphic representation of a distributed execution

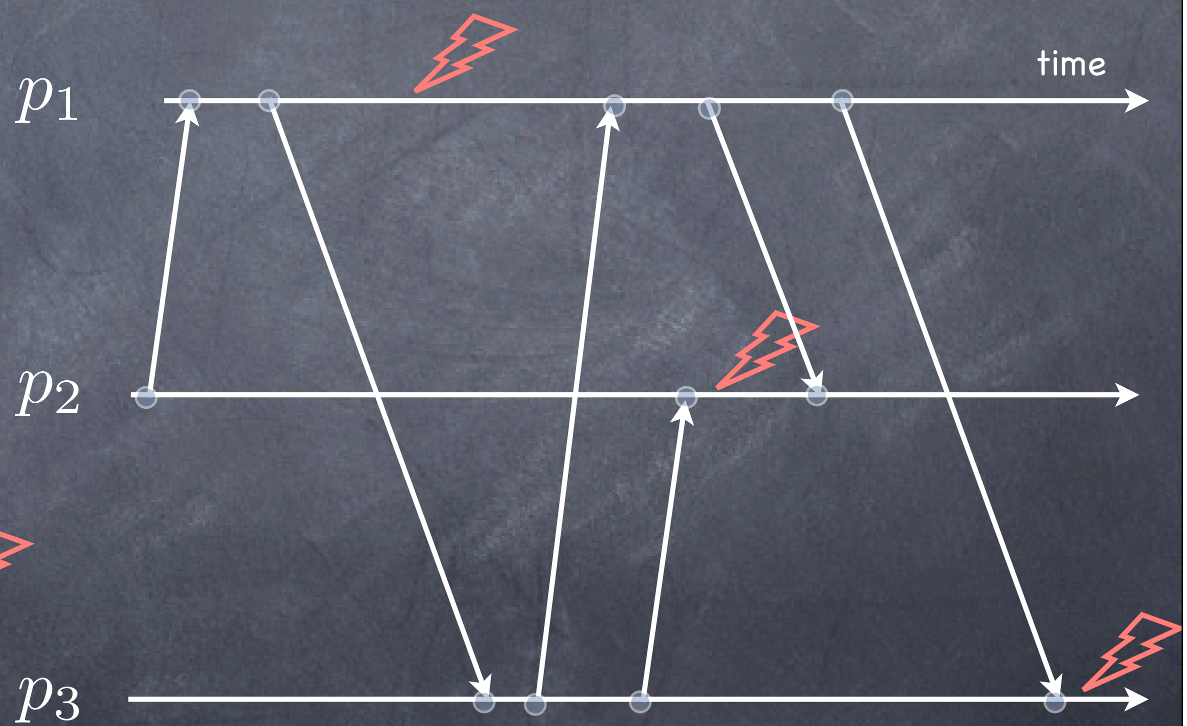




Space-Time diagrams

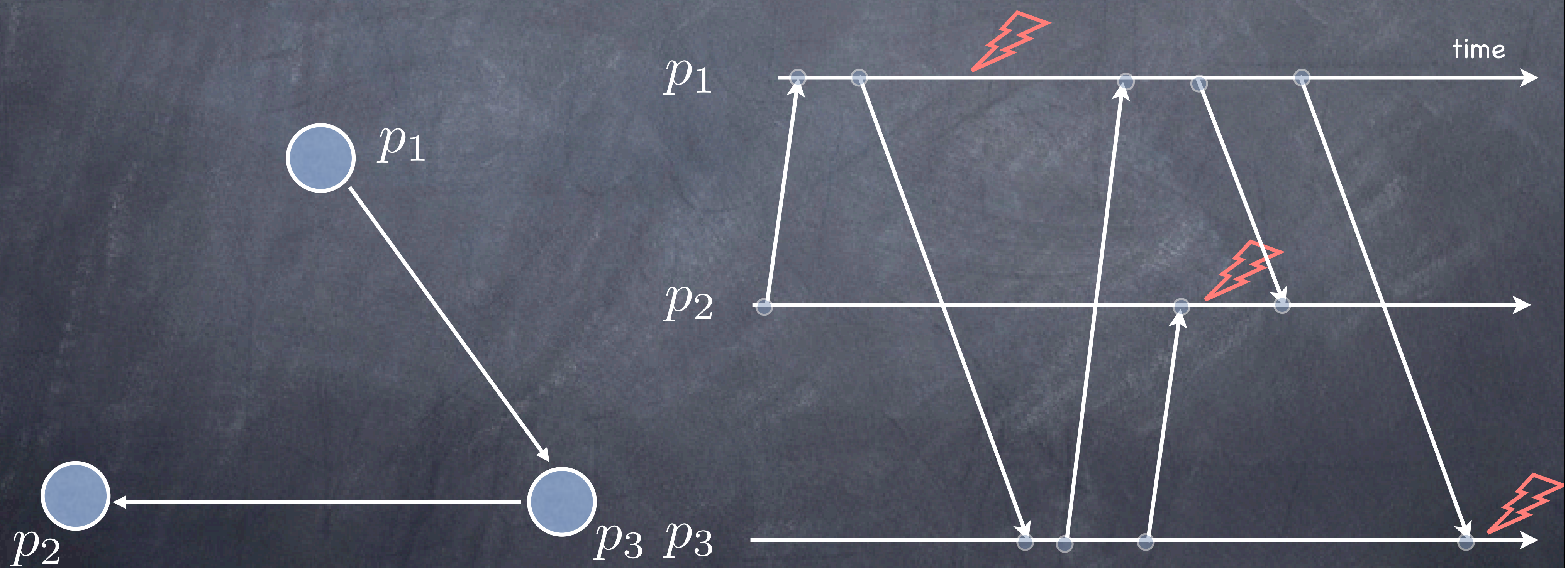
A graphic representation of a distributed execution





Space-Time diagrams

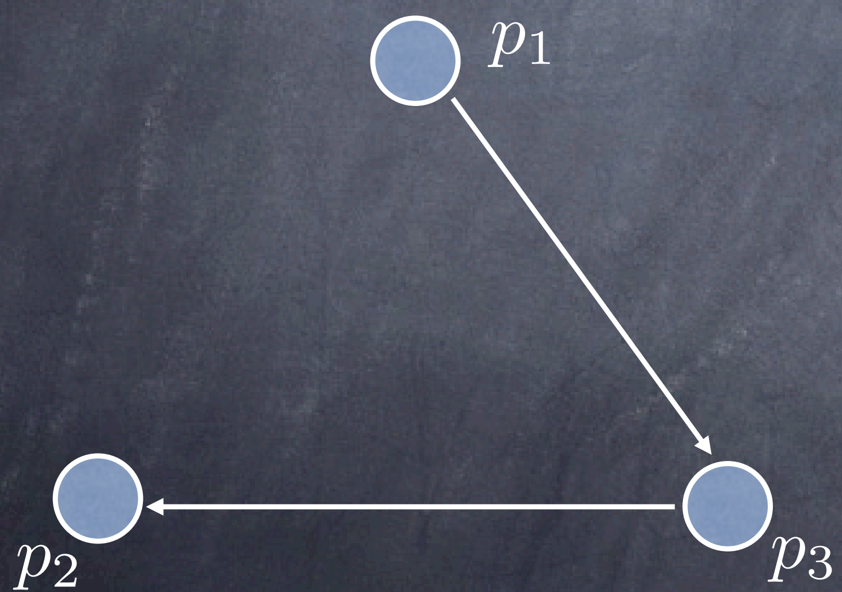
A graphic representation of a distributed execution

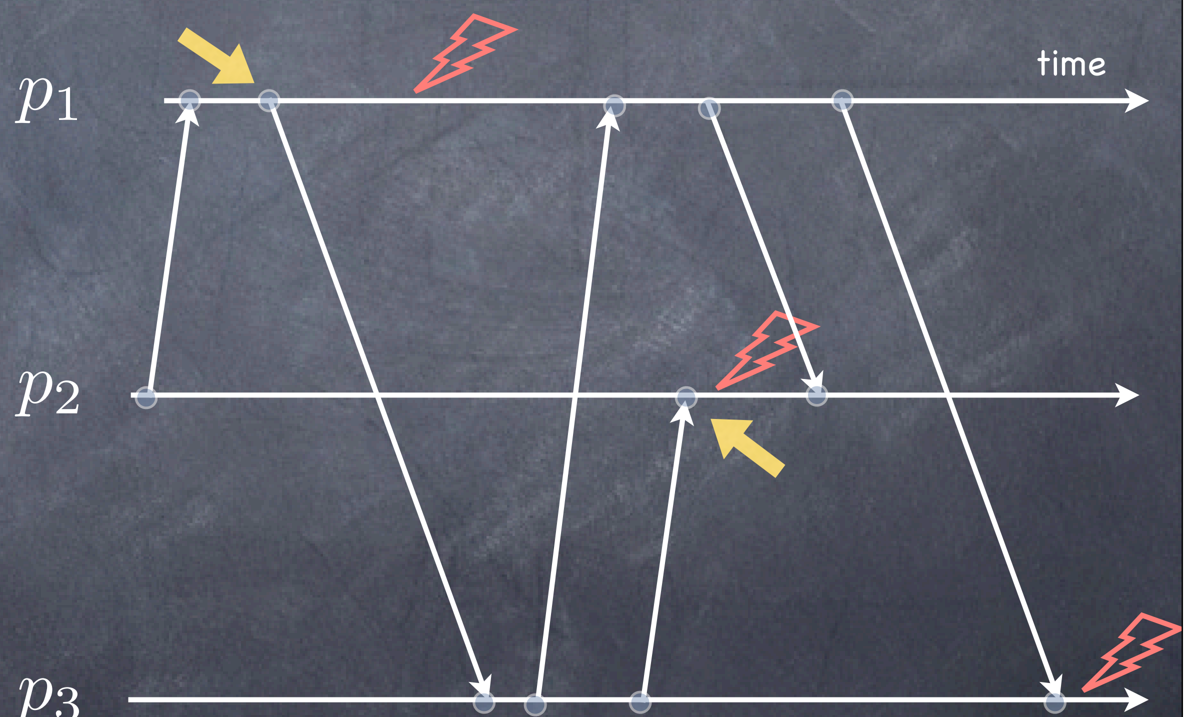


H and $\rightarrow$ impose a partial order

Space-Time diagrams

A graphic representation of a distributed execution

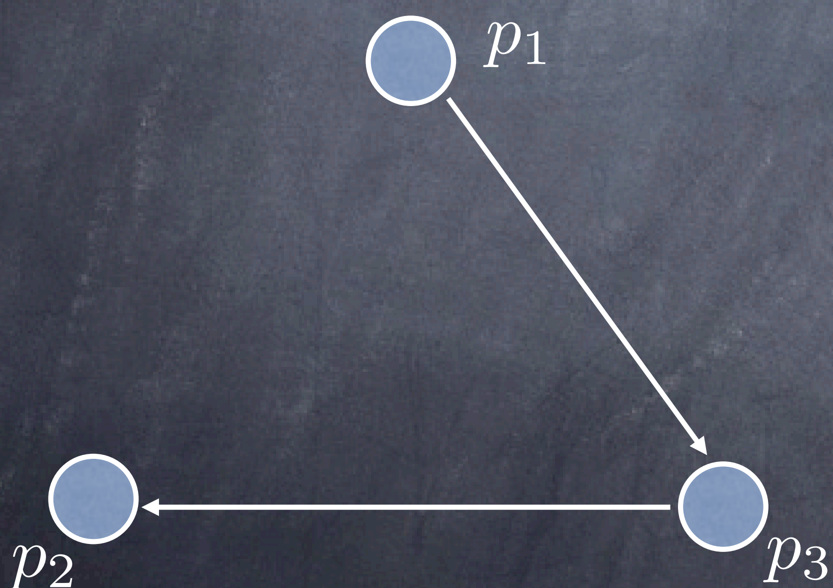


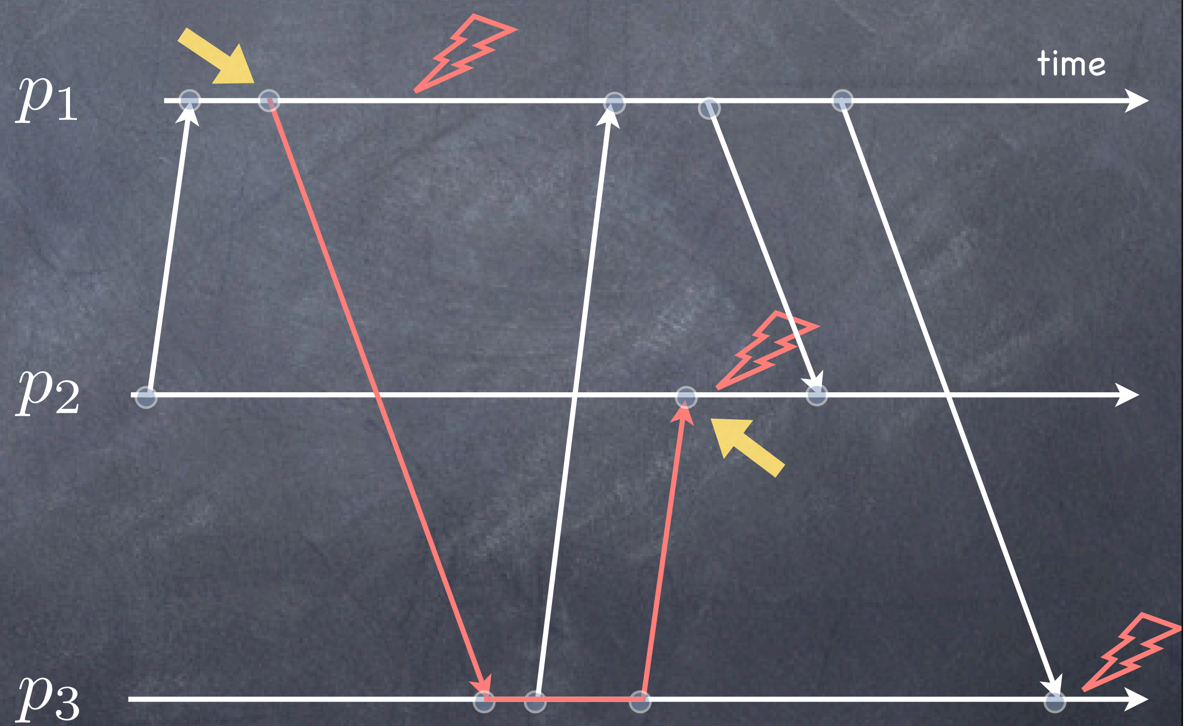


H and $\rightarrow$ impose a partial order

Space-Time diagrams

A graphic representation of a distributed execution

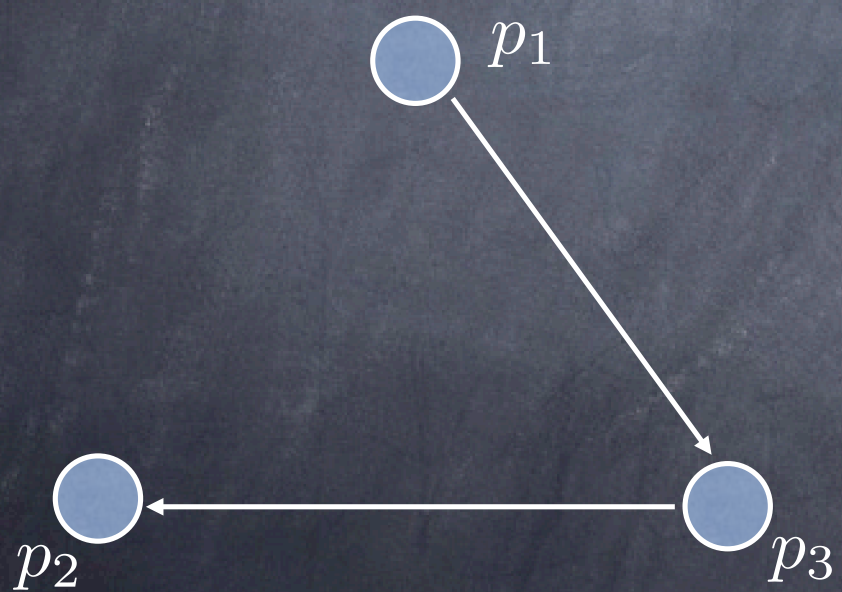


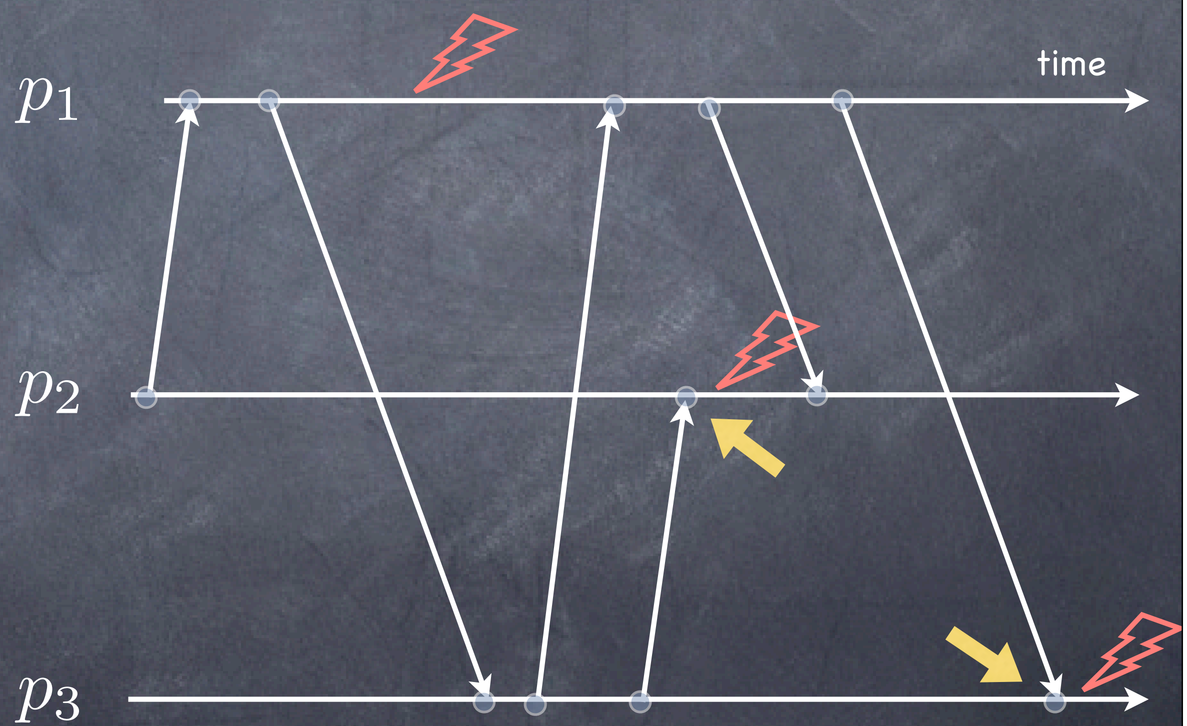


H and $\rightarrow$ impose a partial order

Space-Time diagrams

A graphic representation of a distributed execution





H and $\rightarrow$ impose a partial order

Runs and Consistent Runs

- A run is a total ordering of the events in H that is consistent with the local histories of the processors
  - Ex: $h_1, h_2, \ldots, h_n$ is a run
- A run is consistent if the total order imposed in the run is an extension of the partial order induced by $\rightarrow$
- A single distributed computation may correspond to several consistent runs!